**Practical No.5**

**Name**: Anuj Rajendra Mane

**Div**: SY A

**Roll No**: 65 **Batch**: A3

**PRN** **No**: 2162701242054

**Title: Introduction to Classes & Objects**.

**Theory of class and object:**

A class definition defines a new type of values in Java. It gives us two main capabilities.

First, it allows us to meaningfully group different types of values together to represent some more complex structure, such as a String and an int used to represent a person (for their name and age), or two doubles representing a rectangle (for the length and width). We may call these subvalues by many names: instance variables (because each object is an instance of the class); attributes (because these are the salient parts of any object of the class); field members (a bit more of a traditional name for Java documentation).

Second, we get to define the acceptable behaviors of these values by defining methods ('behaviors'). It gives us the chance to not just represent the world through values, but animate that world-representation through interactions (method calls).

An object represents a specific value of a class. The object has its own value for each attribute listed in the class, and is stored in its own unique place in memory.

When we create a variable to hold an object, we actually store a reference to the object: a 'pointer' into memory that is only allowed to refer to values of a particular type.

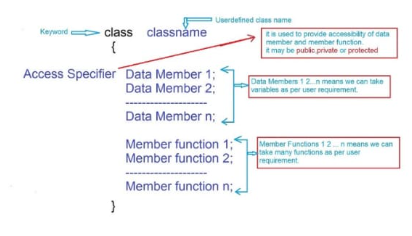
We can create multiple objects from one class. Each object resides in its own location in memory. When we ask a particular object to run a method from its class, the object uses only its own instance variables when running the code.

**Defining a Class**

When we create a class, we can create various members of the class. We can declare variables, to indicate that each value of this newly defined type should have its own value for each of these variables. We can declare methods, each defining the way in which these values can behave.

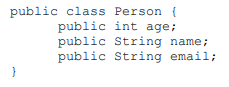
Think of the methods as actions that outsiders can request the object to perform. These actions can be simple, like reporting some quantity or value; they can be complex, like simulating some process, generating new structures, or other tasks like searching/sorting through the instance variables. The statements in a method can be any procedural-style statement – calling other methods, defining variables, using control structures, and performing exception handling. Once we've got a class definition, we can then use it as a recipe to create individual values of this new type, which we call objects. Each object is guaranteed (required) to have its own copy of all the declared variables, and thus we can call all the methods on that object.

**Syntax**:



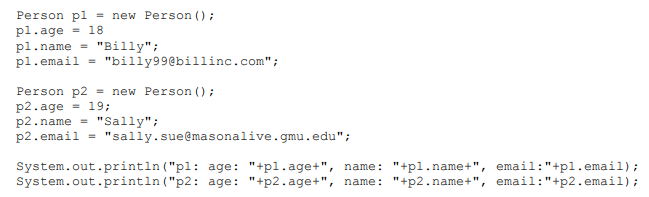
**Class Example**

Here is a very simple example class definition, with no methods:



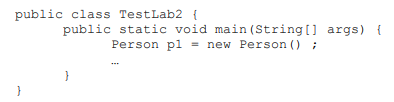
**Creating Objects:**

We could then create Person objects like this:



First of all, where should this code go? A class definition is just a definition, not a specific request for Java to perform any action. A method does nothing, but a method call runs the statements in the method. We learned that running a Java program is synonymous with running the main method of a class, so we want to put this code in a class's main method (or somewhere that will be reached by a class's main method).

We can just add a main method to this class, or we can create an entirely separate class, and add the code to that class's main method. We don't want to get in the habit of thinking the code using a class is required to be in just that class's main method, so let's create a separate class (in the file TestLab2.java):



Now, when we want to run our program, we can just compile TestLab2.java, and run TestLab2, and it will automatically deal with compiling Person.java, and integrating the compiled Person.class file with the TestLab2.class code.



Even though we don't explicitly execute javac Person.java, notice that the Person.class file has been generated. Incidentally, even if the Person class had its own main method, by executing java TestLab2, we are not using Person's main method – we are only executing TestLab2's main method. It is far more common to have many classes involved with only one main method, so that is the usage pattern we are trying to establish.

**Program:**

class Rectangle {

// Data Member

int height;

int width;

int area;

// Member function

void findArea() {

area = height \* width;

System.out.println("Area of Rectangle=" + area);

}

public static void main(String[] args) {

// creating object

Rectangle obj = new Rectangle();

// accessing data member

obj.height = 25;

obj.width = 15;

// accessing member function

obj.findArea();

}

}

**Output**:

**Conclusion:**

I can learn the class and objects in java. In this concept I am able to learn how to create a instance object also learning how to access member and function.

**Completion Date: Co-Ordinator Sign:**